DAA Assignment 6

Group Number - 12

Group Members

- Gitika Yadav - IIT2019219
- Divyatez Singh - IIT2019220
- Divyansh Rai - IIT2019221

Problem

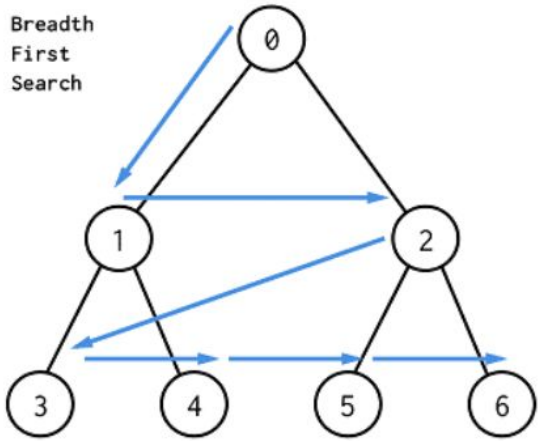
To calculate the shortest distance between any pair location.

Description

To find the minimum distance between any pair location(or cells) in a given 2D matrix(with all 0’s and 1’s, 0 denoting blocked cells).

Breadth First Search

Breadth first search is a general technique of traversing a graph. Breadth first search may use more memory but will always find the shortest path first. It is a path finding algorithm that is capable of always finding the solution if one exists.



Approach

1. Input dimension of given matrix.

2. Store then the matrix and source cell's and destination cell's location.

3. Store each cell as a node with their row, column values and distance from the source cell.

4. Start BFS with the source cell.

5. Make a visited array with all having "false" values.

6. Keep updating distance from source value in each move.

7. Return distance when destination is met, else return -1 (no path exists in between source and destination).

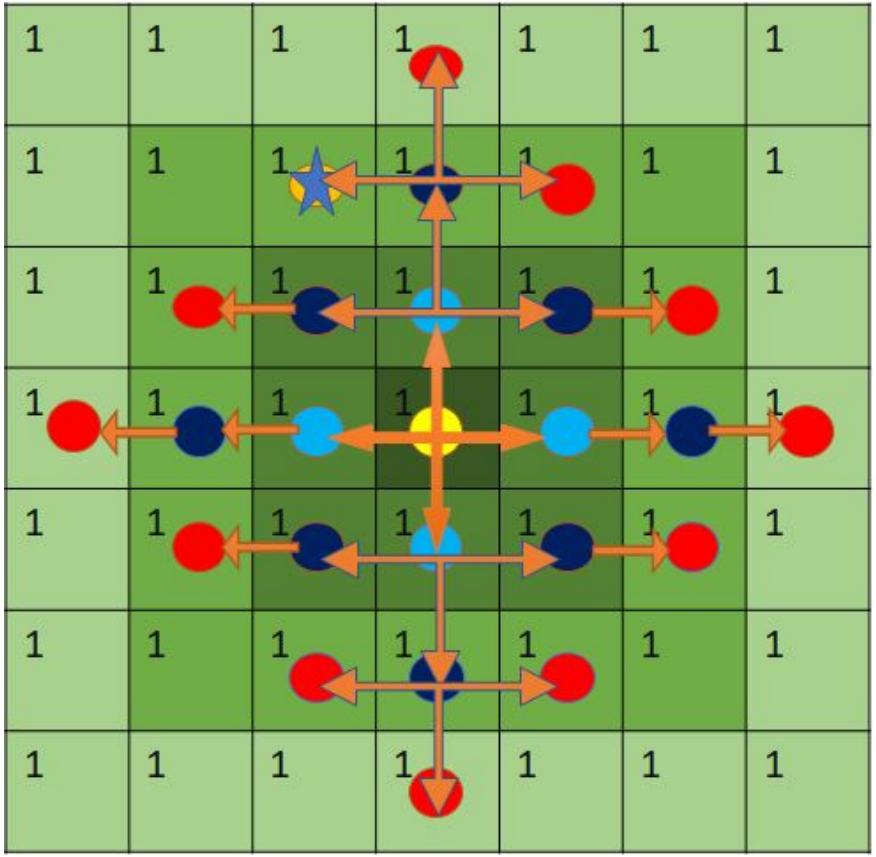
The above returned cost is the sum of distance from source to destination.

Time Complexity

$T(n) = V\ (O(1) + O(Edge\ from\ vertex) + O(1))$

As each vertex can travel in 4 directions:
$T(n) = V + 4V + V$
$T(n) = 2V + E(total\ number\ of\ edges\ in\ graph)$
$T(n) = V + E$

Scanning for all adjacent vertices takes $O(E)$ time, since sum of lengths of adjacency lists is E.
Thus on combining we get the overall time complexity as: $T(n) = O(V+E)$



PSEUDO CODE

```
Int:
procedure main
        n ->row
        m ->column
        a[n][m]->matrix with input
        si,sj ->source cell location
    di,dj ->destination cell location

    if(source cell or destination cell equals 0)
        print( Path  not possible )

    else:
        Initialize queue (of pair of int) q
        Initialize vector vis[n][m] with all
        values initially false

        Initialize cost -> 0
        set vis[si][sj] -> true
        push ({si,sj}) in q

        while(q.size()>0):
                cost++
                p -> q.size

                while(p--):
                    top -> q.front()
                    q.pop()
                    x->top.first
                        y->top.second

            if(x>0):
                    if(valid left cell):
                            Left node visited
                            Queue push left node
            if(y>0):
                    if(valid above cell ):
                            Above node visited
                            Queue push above node
            if(x<n-1):
                    if(valid right cell ):
                            Right node visited
                            Queue push right cell
            if(x>0):
                    if(valid below cell ):
                            Below node visited
                            Queue push below cell

            if(destination cell is visited):
                    print(cost)
                    break
            End of while

            if(destination cell is visited):
                    break
    End of while
    if(destination still not visited):
        print( Not  Found )
```

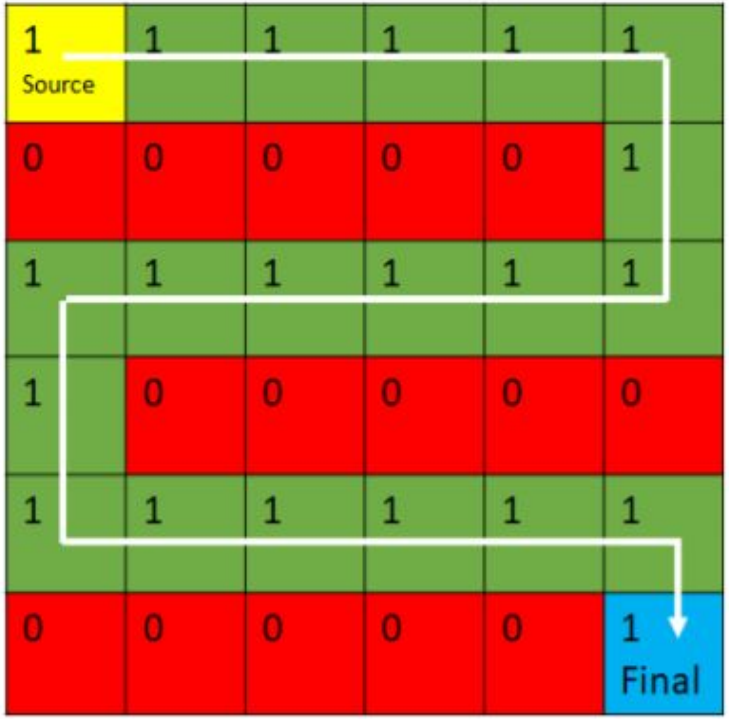


Figure 2: Worst Case Time Complexity

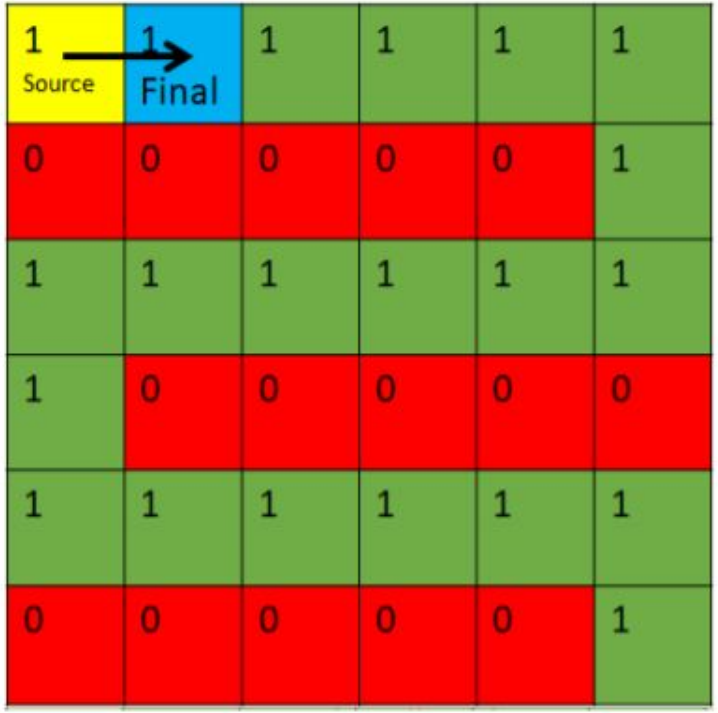
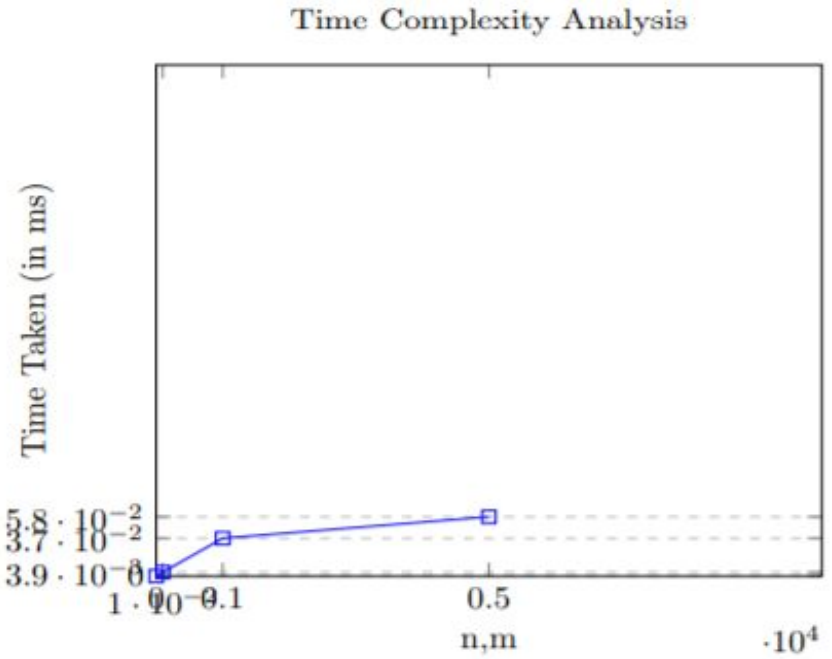
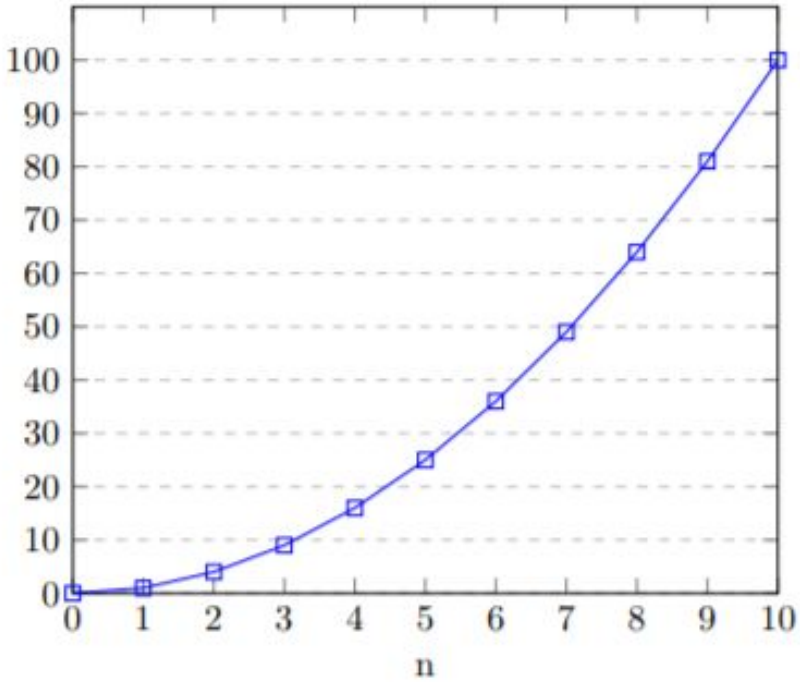


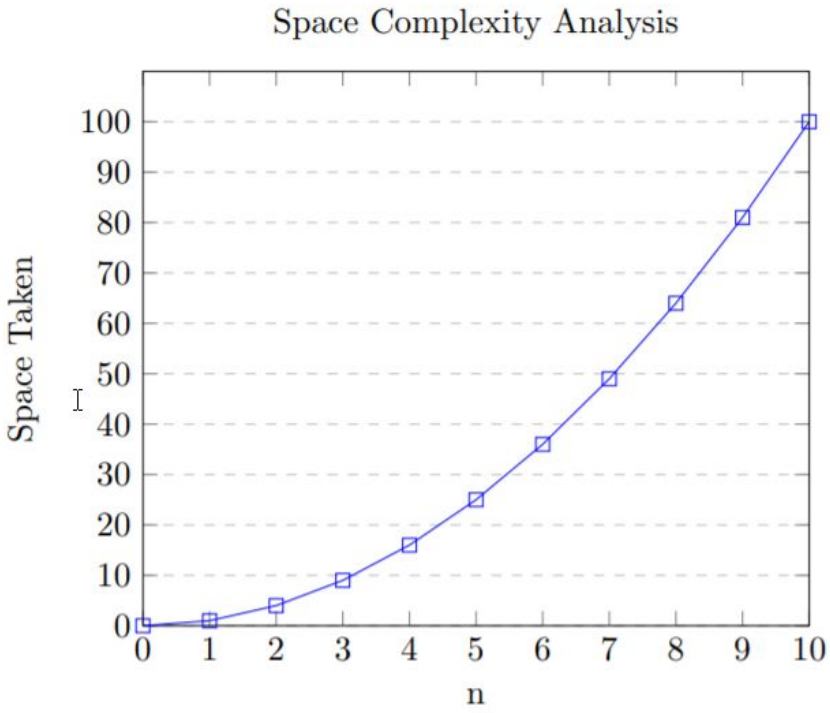
Figure 2: Best Case Time Complexity

Time Complexity





Space Complexity



Thank You